**Algorithmic Thinking in Problem Solving**

**Framework**

1. List the steps one should take when tackling a coding problem during a coding interview. Be very specific. Your task is to create a framework/problem-solving strategy.

Read the whole question, and then take a deep breath in order to not panic, have a reference note that states data structures and simple description since it might forget. After that, solve the problem visualized (drawing) and start the algorithm/pseudocode.

1. What do you think should be done when someone “gets stuck” when solving a coding problem? List the steps. Be very specific. Feel free to use examples to illustrate your points.
2. Take a deep breath
3. Ask question to interviewer
4. Start over
5. Imagine you are being interviewed and you are asked to use a whiteboard to problem solve. How would you use the whiteboard? Draw a diagram that shows the different sections that you would have and justify your decisions.

Draw it out how to solve if need it and then write pseudocode to write a programming to solve the problem.

A picture containing calendar

Description automatically generated

1. Create a checklist for assessing your problem-solving process when tackling a coding problem. For example, one of you items in your checklist could be “Ask Clarifying Questions” or “Test your code”

* Ask clarifying questions
* Test your code
* Did I restate the question
* Did I explain your approach

1. Come up with a list of DOs and DON’Ts that people can use in the context of coding interviews.

Do:

To rest

To use the bathroom before interview

Don’t:

No drinking

1. Use your proposed framework/strategy, whiteboard section distribution, and checklist to tackle the following problems – If you had to refine anything based on your experience tackling the problems, state what you changed and why. A picture containing schematic

   Description automatically generated

Yes, instead of go into pseudocode right away, it is helpful to visualize it first and explain(think) about the algorithm if it solves the problem. And then do the pseudocode. After pseudecode code, it is better to trace it since it may be false algorithm. Afterall, to code it after testing and have confidence that it is right.

Problem A: Given a collection of **distinct** integers, return all possible permutations.

**Example:**

**Input:** [1,2,3]

**Output:**

[

[1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1]

]

Problem B: Given an array of integers nums and a positive integer k, find whether it's possible to divide this array into k non-empty subsets whose sums are all equal.

**Example 1:**

**Input:** nums = [4, 3, 2, 3, 5, 2, 1], k = 4

**Output:** True

**Explanation:** It's possible to divide it into 4 subsets (5), (1, 4), (2,3), (2,3) with equal sums.

**Note:**

* 1 <= k <= len(nums) <= 16.
* 0 < nums[i] < 10000.